Honours Project

Generated by Doxygen 1.9.7

1 Appendix A - Code Documentation

This file contains the documentation of the code for this Project. For information on how to execute the program please see the read me available with the code.

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

4 Data Structure Documentation

4.1 Cell Struct Reference

`#include <fds_types.h>`

Data Fields

- double **c**
- State * **s0**
- State * **s1**
- State * **s2**
- State * **s3**

4.1.1 Detailed Description

A cell in the map. It has a traversal cost c and four states

The documentation for this struct was generated from the following file:

- controllers/uav_controller/src/includes/fds_types.h

4.2 Fds Struct Reference

```
#include <fds.h>
```

Data Fields

- Map * **m**

  A map of the environment.
- State * **start**

  The start node.
- State * **end**

  The goal node.
- HEAP * **OPEN**

  The min heap.

4.2.1 Detailed Description

This struct is used to store the variables used by field D*.

The documentation for this struct was generated from the following file:

- controllers/uav_controller/src/modules/includes/fds.h

4.3 Follower Struct Reference

The Follower structure.

```
#include <uav.h>
```

Data Fields

- unsigned char **id**

  The id of the follower.
- Vec3d **pos**

  The initial position of the follower.
- int **wp_num**

  The number of the waypoint the follower is currently following.
- Vec3d * **wps**

  List of waypoints computed for the follower.

4.3.1 Detailed Description

The Follower structure.

The documentation for this struct was generated from the following file:

- controllers/uav_controller/src/includes/uav.h

4.4 HEAP Struct Reference

`#include <heap.h>`

Data Fields

- NODE * **arr**
- int **arr_len**
- int **h_len**
- char **type**

4.4.1 Detailed Description

The heap strucure containing the array and lenght information as well as the type

The documentation for this struct was generated from the following file:

- controllers/uav_controller/src/util/includes/heap.h

4.5 MData Union Reference

`#include <net.h>`

Data Fields

- struct {
  double **x**
  double **y**
  };
- int **wp_num**
- unsigned char **bytes** [MD_SIZE]

4.5.1 Detailed Description

The data part of the message. It accepst either a set of coordinates or an integer as the waypoint number

The documentation for this union was generated from the following file:

- controllers/uav_controller/src/modules/includes/net.h

4.6 Message Union Reference

`#include <net.h>`

Data Fields

- struct {
  MHead **head**
  unsigned char **padding** [5]
  MData **data**
  };
- unsigned char **bytes** [M_SIZE]

4.6.1 Detailed Description

The full message union

The documentation for this union was generated from the following file:

- controllers/uav_controller/src/modules/includes/net.h

4.7 MHead Union Reference

`#include <net.h>`

Data Fields

- struct {
  unsigned char **s_id**
  The sender's id.
  unsigned char **r_id**
  The receiver's id.
  unsigned char **type**
  The type of message. See MType enum.
  };
- unsigned char **bytes** [MH_SIZE]

4.7.1 Detailed Description

The header of the message

The documentation for this union was generated from the following file:

- controllers/uav_controller/src/modules/includes/net.h

4.8 NODE Struct Reference

`#include <heap.h>`

Data Fields

- void * **key**
- void * **val**

4.8.1 Detailed Description

The node structure of the heap. It contains one element of the heap, its value and its key

The documentation for this struct was generated from the following file:

- controllers/uav_controller/src/util/includes/heap.h

4.9 Position Struct Reference

Used for storing position and attitude informations.

`#include <uav.h>`

Data Fields

- double **x**
- double **y**
- double **z**
- double **pitch**
- double **roll**
- double **yaw**

4.9.1 Detailed Description

Used for storing position and attitude informations.

The documentation for this struct was generated from the following file:

- controllers/uav_controller/src/includes/uav.h

4.10 State Struct Reference

`#include <fds_types.h>`

Data Fields

- Vec3d **v**

  The position of the State defined by a vector v.
- double **rhs**

  The one-step lookahead value of the state.
- double **g**

  The g-value of the state. Represents the path cost of this state.
- char **visited**

  Equals one if the state is visited, otherwise it will be zero.
- struct State * **s1**

  Used for path extraction.
- struct State * **s2**

  Used for path extraction.

4.10.1 Detailed Description

The state structure is usde to represent a vertex in Field D* and is used to compute the shortest path

The documentation for this struct was generated from the following file:

- controllers/uav_controller/src/includes/fds_types.h

4.11 Tuple Struct Reference

Data Fields

- State * **fst**
- State * **snd**

The documentation for this struct was generated from the following file:

- controllers/uav_controller/src/util/includes/map.h

4.12 uav Struct Reference

The UAV structure containing the UAV data.

`#include <uav.h>`

Data Fields

- WbDeviceTag **camera**

  The camera device tag (unused)
- WbDeviceTag **imu**

  The inertial reference unit device tag.
- WbDeviceTag **gps**

  The gps device tag.
- WbDeviceTag **compass**

  The compass device tag.
- WbDeviceTag **gyro**

  The gyro device tag.
- WbDeviceTag **emitter**

  The emitter device tag.
- WbDeviceTag **receiver**

  The receiver device tag.
- WbDeviceTag **radar**

  The radar device tag.
- WbDeviceTag **front_left_led**

  The front left led device tag (unused)
- WbDeviceTag **front_right_led**

  The front right led device tag (unused)
- WbDeviceTag **camera_roll_motor**

  The camera roll motor device tag (unused)
- WbDeviceTag **camera_pitch_motor**

  The camera pitch motor device tag (unused)
- WbDeviceTag **motors** [4]

  The quadcopter motors device tag.
- double **target_alt**

  Target altitude.
- Position **pos**

  The UAV current position. It also stores the registered attitude.

- double **t**

  Time mesured in ms.
- double **pitch_disturbance**

  Computed pitch disturbance.
- double **yaw_disturbance**

  Computed yaw disturbance.
- int **target_reached**

  Traged reached flag.
- Fds * **fds**

  Pointer to the field D global variables.*
- char **state**

  The current state the UAV is in. Used to control the program.
- unsigned char **id**

  The id of the UAV. Used for inter-UAV connection.
- unsigned char **l_id**

  The id of the elected leader. If id == l_id then UAV is leader.
- Follower * **followers**

  An array of all detected followers.
- int **f_num**

  The number of followers in the array.

4.12.1 Detailed Description

The UAV structure containing the UAV data.

This data structure contains all device tags to interface with the Webots robot. It also contains other varaibles needed in multiple functions

The documentation for this struct was generated from the following file:

- controllers/uav_controller/src/includes/uav.h

4.13 Vec3d Union Reference

`#include <vec.h>`

Data Fields

- struct {
  double **x**
  double **y**
  double **z**
  };

- struct {
  double **pitch**
  double **roll**
  double **yaw**
  };

4.13.1 Detailed Description

A three-dimensional vector

The documentation for this union was generated from the following file:

- controllers/uav_controller/src/util/includes/vec.h

5 File Documentation

5.1 controllers/uav_controller/src/includes/fds_types.h File Reference

```
#include ¨../util/includes/vec.h¨
```

Data Structures

- struct State
- struct Cell

Typedefs

- typedef struct State State

5.1.1 Detailed Description

This file contains types used by Field D∗

5.1.2 Typedef Documentation

State

```
typedef struct State State
```

The state structure is usde to represent a vertex in Field D∗ and is used to compute the shortest path

5.2 fds_types.h

Go to the documentation of this file.

```
00001
00007 #include "../util/includes/vec.h"
00008
00009 #ifndef FDS_TYPES_H
00010 #define FDS_TYPES_H
00011
00017 typedef struct State {
00018     Vec3d v;
00019     double rhs;
00020     double g;
00021     char visited;
00022     struct State *s1;
00023     struct State *s2;
00024 } State;
00025
00030 typedef struct {
00031     double c;
00032     State* s0;
00033     State* s1;
00034     State* s2;
00035     State* s3;
00036 } Cell;
00037
00038 #endif // !FDS_TYPES_H
```

5.3 controllers/uav_controller/src/includes/uav.h File Reference

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <webots/robot.h>
#include <webots/compass.h>
#include <webots/gps.h>
#include <webots/gyro.h>
#include <webots/inertial_unit.h>
#include <webots/keyboard.h>
#include <webots/led.h>
#include <webots/motor.h>
#include <webots/camera.h>
#include <webots/emitter.h>
#include <webots/receiver.h>
#include <webots/radar.h>
#include ¨../util/includes/util.h¨
#include ¨../modules/includes/comm_module.h¨
#include ¨../modules/includes/fds.h¨
#include ¨../modules/includes/net.h¨
```

Data Structures

- struct Position

  Used for storing position and attitude informations.
- struct Follower

  The Follower structure.
- struct uav

  The UAV structure containing the UAV data.

Macros

- #define ROLL_P 50.0

  libc includes
- #define **PITCH_P** 30.0

  Pitch constant for PID.
- #define **VERTICAL_P** 3.0

  Vertical constant for PID.
- #define **VERTICAL_T** 68.5

  Vertical constant for PID.
- #define **VERTICAL_O** 0.6

  Vertical constant for PID.
- #define **TARGET_PRECISION** 0.5

  This constant determines how close the UAV must be to the waypoint for it to be reaced.
- #define **MAX_YAW_DIST** 0.4

  Maximum Yaw Displacement.
- #define **MAX_PITCH_DIST** -1

  Maximum Pitch Displacement.

Typedefs

- typedef struct Position **Position**

  Used for storing position and attitude informations.
- typedef struct uav Uav

  The UAV structure containing the UAV data.

Enumerations

- enum Uav_State {
  INIT , RUN , END , F_RUN ,
  F_WAIT }

  The Possible states of the UAV.

Functions

- void **uav_init** (Uav *uav, int timestep)

  UAV initialization function.
- double **uav_get_roll** (Uav *uav)

  Returns the current roll.
- double **uav_get_pitch** (Uav *uav)

  Returns the current pitch.
- double **uav_get_yaw** (Uav *uav)

  Returns the current yaw.
- double **uav_get_gps_altitude** (Uav *uav)

  Returns the current registered altitude.
- const double * **uav_get_gps_pos** (Uav *uav)

  Returns the current gps position.
- double **uav_get_roll_velocity** (Uav *uav)

  Returns the roll velocity.

- double **uav_get_pitch_velocity** (Uav *uav)

  Returns the pitch velocity.
- double **uav_get_yaw_velocity** (Uav *uav)

  Returns the yaw velocity.
- double **uav_get_heading** (Uav *uav)

  Returns the current heading in degrees.
- int **uav_get_radar_targets_num** (Uav *uav)

  Returns the number of trargets.
- const WbRadarTarget * **uav_get_radar_targets** (Uav *uav)

  Returns an array of tragets.
- int **uav_get_obstacles** (Uav *uav, WbRadarTarget *targets)

  Given an empty array of targets, this function will return the numebr of targets and the targets themselves.
- void **uav_set_position** (Uav *uav, Position position)

  Sets the position of the UAV.
- void **uav_actuate_motors** (Uav *uav, double roll, double pitch, double yaw, double altitude)

  Actuates the motors of the UAV.
- void **uav_wait** (int timestep, double x)

  Waits for x seconds.
- void **uav_cleanup** (Uav *uav)

  Cleanup function.
- void **uav_send_msg** (Uav *uav, const Message m)

  Send a message m.
- Message **uav_receive_msg** (Uav *uav, int *queue_len)

  Receive a message and get the number of messages in the queue.
- int **uav_get_msg_num** (Uav *uav)

  Returns the number of messages currently on the queue.
- int **uav_peek_msg** (Uav *uav)

  Peek at next message but do not remove from queue.
- int **cm_run** (Uav *uav, Vec3d wp, double target_alt, double time)

  Runs the automated movement of the uav.
- Vec3d * **cm_detect_obstacles** (Uav *uav, int *num)

  Detects the obstacles and returns the number and the position of the obstacles.
- Vec3d * **cm_plan_path** (Uav *uav, int *wps_num)

  Plans the path using Filed D.*
- void **cm_followers_path** (Uav *uav, Vec3d *wps, int wps_num)

  Plans the path of the followers.
- void **net_elect_leader** (Uav *uav, int timestep)

  Leader election function.
- void **net_share_init_pos** (Uav *uav, int timestep)

  Sends inital position to leader. If the calling UAV is the leader then it will receive the followers psoitions.
- void **net_send_wp** (Uav *uav, int curr_wp)

  Send the current waypoint to the follower.
- void **net_recieve_wp** (Uav *uav, Vec3d *wp)

  Receive waypoint from leader.
- void **net_ask_next_wp** (Uav *uav, int curr_wp)

  Ask next waypoint from leader (unused)
- void **fds_cleanup** (Fds *fds)

  Cleanup fds structure.

5.3.1 Detailed Description

This file contains most strucutres and function definition for the uav and the command module

5.3.2 Macro Definition Documentation

ROLL_P

```
#define ROLL_P 50.0
```

libc includes

webots robot includes webots devices includes local header files Roll costant for PID

5.3.3 Typedef Documentation

Uav

```
typedef struct uav Uav
```

The UAV structure containing the UAV data.

This data structure contains all device tags to interface with the Webots robot. It also contains other varaibles needed in multiple functions

5.3.4 Enumeration Type Documentation

Uav_State

```
enum Uav_State
```

The Possible states of the UAV.

Enumerator

INIT	Set the UAV to intialization phase.
RUN	Set the UAV to run phase and declares the UAV as leader.
END	Set the UAV to cleanup phase and terminate the program.
F_RUN	Set the UAV to run pahse as a follower.
F_WAIT	Set the UAV to wait phase and wait for leader's instructions.

5.4 uav.h

Go to the documentation of this file.

```
00001
00008 #ifndef UAV_H
```

```
00009 #define UAV_H
00010
00012 #include <math.h>
00013 #include <stdio.h>
00014 #include <stdlib.h>
00015
00017 #include <webots/robot.h>
00018
00020 #include <webots/compass.h>
00021 #include <webots/gps.h>
00022 #include <webots/gyro.h>
00023 #include <webots/inertial_unit.h>
00024 #include <webots/keyboard.h>
00025 #include <webots/led.h>
00026 #include <webots/motor.h>
00027 #include <webots/camera.h>
00028 #include <webots/emitter.h>
00029 #include <webots/receiver.h>
00030 #include <webots/radar.h>
00031
00033 #include "../util/includes/util.h"
00034 #include "../modules/includes/comm_module.h"
00035 #include "../modules/includes/fds.h"
00036 #include "../modules/includes/net.h"
00037
00038 // Uncomment for debug functionality
00039 //#define DEBUG
00040
00041 // Constants
00042 #define ROLL_P 50.0
00043 #define PITCH_P 30.0
00044 #define VERTICAL_P 3.0
00045 #define VERTICAL_T 68.5
00046 #define VERTICAL_O 0.6
00047 #define TARGET_PRECISION 0.5
00048 #define MAX_YAW_DIST 0.4
00049 #define MAX_PITCH_DIST -1
00050
00052 typedef struct Position {
00053     double x;
00054     double y;
00055     double z;
00056     double pitch;
00057     double roll;
00058     double yaw;
00059 } Position;
00060
00062 enum Uav_State {
00063     INIT,
00064     RUN,
00065     END,
00066     F_RUN,
00067     F_WAIT,
00068 };
00069
00071 typedef struct {
00072     unsigned char id;
00073     Vec3d pos;
00074     int wp_num;
00075     Vec3d *wps;
00076 } Follower;
00077
00079
00083 typedef struct uav {
00084     /* Devices */
00085     WbDeviceTag camera;
00086     WbDeviceTag imu;
00087     WbDeviceTag gps;
00088     WbDeviceTag compass;
00089     WbDeviceTag gyro;
00090     WbDeviceTag emitter;
00091     WbDeviceTag receiver;
00092     WbDeviceTag radar;
00093
00094     /* Led Lights */
00095     WbDeviceTag front_left_led;
00096     WbDeviceTag front_right_led;
00097
00098     /* Motors */
00099     WbDeviceTag camera_roll_motor;
00100     WbDeviceTag camera_pitch_motor;
00101
00102     WbDeviceTag motors[4];
00103
00104     /* Variables */
00105     double target_alt;
00106     Position pos;
```

```
00107
00108     double t;
00109
00110     double pitch_disturbance;
00111     double yaw_disturbance;
00112
00113     int target_reached;
00114
00115     /* Path planning */
00116     Fds *fds;
00117
00118     /* State */
00119     char state;
00120
00121     /* Networking */
00122     unsigned char id;
00123     unsigned char l_id;
00124
00125     Follower *followers;
00126     int f_num;
00127 } Uav;
00128
00130 void uav_init(Uav* uav, int timestep);
00131
00132 /* Getters */
00133 double uav_get_roll(Uav* uav);
00134 double uav_get_pitch(Uav* uav);
00135 double uav_get_yaw(Uav* uav);
00136
00137 double uav_get_gps_altitude(Uav* uav);
00138 const double* uav_get_gps_pos(Uav* uav);
00139
00140 double uav_get_roll_velocity(Uav* uav);
00141 double uav_get_pitch_velocity(Uav* uav);
00142 double uav_get_yaw_velocity(Uav* uav);
00143
00144 double uav_get_heading(Uav* uav);
00145
00146 int uav_get_radar_targets_num(Uav* uav);
00147 const WbRadarTarget* uav_get_radar_targets(Uav* uav);
00148 int uav_get_obstacles(Uav* uav, WbRadarTarget* targets);
00149
00150 /* Setters */
00151 void uav_set_position(Uav* uav, Position position);
00152
00153 /* Other methods */
00154 void uav_actuate_motors(Uav* uav, double roll, double pitch, double yaw, double altitude);
00155 void uav_wait(int timestep, double x);
00156 void uav_cleanup(Uav *uav);
00157
00158 /* Message passing functons */
00159 void uav_send_msg(Uav *uav, const Message m);
00160 Message uav_receive_msg(Uav *uav, int *queue_len);
00161 int uav_get_msg_num(Uav *uav);
00162 int uav_peek_msg(Uav *uav);
00163
00164 /* Control Module Functions */
00165 int cm_run(Uav *uav, Vec3d wp, double target_alt, double time);
00166 Vec3d* cm_detect_obstacles(Uav *uav, int *num);
00167 Vec3d* cm_plan_path(Uav *uav, int *wps_num);
00168 void cm_followers_path(Uav *uav, Vec3d *wps, int wps_num);
00169
00170 /* Network function */
00171 void net_elect_leader(Uav *uav, int timestep);
00172 void net_share_init_pos(Uav *uav, int timestep);
00173 void net_send_wp(Uav *uav, int curr_wp);
00174 void net_recieve_wp(Uav *uav, Vec3d *wp);
00175 void net_ask_next_wp(Uav *uav, int curr_wp);
00176
00177 /* Field D* functions */
00178 void fds_cleanup(Fds *fds);
00179
00180 #endif // !UAV_H
```

5.5 controllers/uav_controller/src/main.c File Reference

Main file of the controller.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
```

```
#include ¨includes/uav.h¨
#include ¨modules/includes/fds.h¨
#include ¨util/includes/map.h¨
#include ¨util/includes/util.h¨
#include ¨util/includes/vec.h¨
```

Macros

- #define **GOAL_X** 20.0f
- #define **GOAL_Y** 0.0f
- #define **TARGT_ALT** 2.0f

Functions

- void set_id (Uav ∗uav, Vec3d start)
- void set_start_and_goal (Uav ∗uav)
- int init ()
- void run ()
- void follower_wait ()
- void follower_run ()
- void main_loop (int timestep)
- void clean_up ()
- int main (int argc, char ∗argv[ ])

Variables

- Uav **uav**

5.5.1 Detailed Description

Main file of the controller.

This file contains the initialization, main loop and clean up functions.

5.5.2 Function Documentation

clean_up()

```
void clean_up ( )
```

The clean up function

follower_run()

```
void follower_run ( )
```

The main function of the followers. It handles the received waypoints and coordinates the movements.

follower_wait()

```
void follower_wait ( )
```

This functions makes the followers wait until a message is received

init()

```
int init ( )
```

This function initializes the program. First it initializes the webots controller, then it initializes debug logging, and the UAV itself. It also call a function to set the start goal.

main()

```
int main (
            int argc,
            char * argv[] )
```

The main function

main_loop()

```
void main_loop (
            int timestep )
```

The main loop function proper. Depending on the values of the state of the UAV different functions are called.

run()

```
void run ( )
```

This function is the backbone of the program. It is the main function of the leader UAV and it is responisble for planning the path of the leader and followers, and coordinate the movements.

set_id()

```
void set_id (
            Uav * uav,
            Vec3d start )
```

This function randomly generates an id give the initial staring position of the UAV. It uses the position as seeds.

set_start_and_goal()

```
void set_start_and_goal (
            Uav * uav )
```

This funtion gets the GPS position of the UAV and creates two vector, goal and start, based on predefined value and the GPS position. The initial position is rounded to the nearest integer.

5.6 controllers/uav_controller/src/modules/includes/fds.h File Reference

```
#include ¨../../includes/fds_types.h¨
#include ¨../../util/includes/map.h¨
#include ¨../../util/includes/heap.h¨
```

Data Structures

- struct Fds

Functions

- char **compare_keys** (void ∗k1, void ∗k2)

  This functions compares two keys together. Returns 1 if k1 is less than k2 and 0 otherwise.
- void **UpdateState** (Fds ∗fds, State ∗s)

  Updates the state passed in as argument.
- void **ComputeShortestPath** (Fds ∗fds)

  Finds the shortest path using field D∗.
- Fds ∗ **fds_init** (Vec3d start, Vec3d goal)

  Initializes the algorithm.
- void **fds_run** (Fds ∗fds, Cell ∗∗changed_cells, int num_cells)

  Executes the main loop of the algorithm.
- Vec3d ∗ **fds_extract_path** (Fds ∗fds, int ∗vec_num)

  Extracts the path from the map after filed D∗ was executed.

5.6.1 Detailed Description

This is the header file for the field D∗ algorithm implementation

5.7 fds.h

Go to the documentation of this file.

```
00001
00006 #ifndef FDS_H
00007 #define FDS_H
00008
00009 #include "../../includes/fds_types.h"
00010 #include "../../util/includes/map.h"
00011 #include "../../util/includes/heap.h"
00012
00017 typedef struct {
00018     Map *m;
00019     State *start;
00020     State *end;
00021     HEAP *OPEN;
00022 } Fds;
00023
00024 char compare_keys(void* k1, void* k2);
00025 void UpdateState(Fds *fds, State *s);
00026 void ComputeShortestPath(Fds *fds);
00027
00028 Fds* fds_init(Vec3d start, Vec3d goal);
00029 void fds_run(Fds* fds, Cell** changed_cells, int num_cells);
00030 Vec3d* fds_extract_path(Fds* fds, int *vec_num);
00031
00032
00033 #endif // !FDS_H
```

5.8 controllers/uav_controller/src/modules/includes/net.h File Reference

Data Structures

- union MHead
- union MData
- union Message

Macros

- #define **MH_SIZE** 3

  The size of the head of the message in bytes.
- #define **MD_SIZE** sizeof(double) ∗ 2

  The size of the data of the message in bytes.
- #define **M_SIZE** MH_SIZE + MD_SIZE + 5

  The full size of a message with 5 bytes of padding after the head.

Enumerations

- enum MType { ID , POS , WP , NEXT_WP }

5.8.1 Detailed Description

This is the header file for most networking functionalities. It contains the message structure.

5.8.2 Enumeration Type Documentation

MType

```
enum MType
```

The enum of the message type. Possible values are ID, POS, WP and NEXT_WP

Enumerator

ID	Used to send ids.
POS	Used to share position.
WP	Used for sending and receiving waypoints.
NEXT_WP	Used to request next waypoint.

5.9 net.h

Go to the documentation of this file.

```
00001
00007 #ifndef NET_H
00008 #define NET_H
```

```
00009
00010 #define MH_SIZE 3
00011 #define MD_SIZE sizeof(double) * 2
00012 #define M_SIZE MH_SIZE + MD_SIZE + 5
00013
00018 enum MType {
00019     ID,
00020     POS,
00021     WP,
00022     NEXT_WP
00023 };
00024
00028 typedef union {
00029     struct {
00030         unsigned char s_id;
00031         unsigned char r_id;
00032         unsigned char type;
00033     };
00034     unsigned char bytes[MH_SIZE];
00035 } MHead;
00036
00042 typedef union {
00043     struct {
00044         double x;
00045         double y;
00046     };
00047     int wp_num;
00048     unsigned char bytes[MD_SIZE];
00049 } MData;
00050
00054 typedef union {
00055     struct {
00056         MHead head;
00057         unsigned char padding[5];
00058         MData data;
00059     };
00060     unsigned char bytes[M_SIZE];
00061 } Message;
00062
00063 #endif // !NET_H
```

5.10 controllers/uav_controller/src/util/includes/heap.h File Reference

Data Structures

- struct NODE
- struct HEAP

Macros

- #define **MIN_HEAP** 0

  Use to set the heap to min heap.
- #define **MAX_HEAP** 1

  Use to set the heap to max heap.
- #define **get_root_val**(h) h->arr[0].val

  Returns the root value of the heap.
- #define **get_root_key**(h) h->arr[0].key

  Returns the root key of the heap.
- #define **pop_root_val**(h) heap_extract(h).val

  Pops the root value of the heap. It removes the root element.
- #define **pop_root_key**(h) heap_extract(h).key

  Pops the root key of the heap. It removes the root element.
- #define **pop_root**(h) heap_extract(h)

  Pops the root element of the heap.

Typedefs

- typedef struct NODE NODE
- typedef struct HEAP HEAP

Functions

- NODE **heap_extract** (HEAP ∗h)

  Extract the root value from the heap.
- HEAP ∗ new_heap (void ∗data, void ∗∗keys, int h_len, char(∗k_comp)(void ∗, void ∗), char(∗v_comp)(void ∗, void ∗), void ∗max_val)
- int **heap_add** (HEAP ∗h, void ∗val, void ∗key)

  Add a new value with a specific key to the heap.
- void **heap_destroy** (HEAP ∗h)

  Destroy the heap.
- char **heap_remove** (HEAP ∗h, void ∗val)

  Remove a specific value from the heap or do nothing if the value is not there.

5.10.1 Detailed Description

Header file of the heap

5.10.2 Typedef Documentation

HEAP

```
typedef struct HEAP HEAP
```

The heap strucure containing the array and lenght information as well as the type

NODE

```
typedef struct NODE NODE
```

The node structure of the heap. It contains one element of the heap, its value and its key

5.10.3 Function Documentation

new_heap()

```
HEAP * new_heap (
            void * data,
            void ** keys,
            int h_len,
            char(*)(void *, void *) k_comp,
            char(*)(void *, void *) v_comp,
            void * max_val )
```

Creates a new heap and return a pointer to it. This function requries as arguments an array of elements and keys, both arrays can be empty, the length of the passed data, a comparsion function for both keys and values and the maximum allowed value

5.11 heap.h

Go to the documentation of this file.

```
00001
00006 #ifndef HEAP_H
00007 #define HEAP_H
00008
00009 #define MIN_HEAP 0
00010 #define MAX_HEAP 1
00011
00012 #define get_root_val(h) h->arr[0].val
00013 #define get_root_key(h) h->arr[0].key
00014
00015 #define pop_root_val(h) heap_extract(h).val
00016 #define pop_root_key(h) heap_extract(h).key
00017 #define pop_root(h) heap_extract(h)
00018
00024 typedef struct NODE {
00025     void* key;
00026     void* val;
00027 } NODE;
00028
00033 typedef struct HEAP {
00034     NODE *arr;
00035     int arr_len;
00036     int h_len;
00037     char type;
00038 } HEAP;
00039
00040 NODE heap_extract(HEAP* h);
00041
00049 HEAP* new_heap(void *data, void **keys, int h_len, char (*k_comp)(void*, void*), char (*v_comp)(void*,
      void*), void* max_val);
00050
00051 int heap_add(HEAP* h, void* val, void *key);
00052 void heap_destroy(HEAP* h);
00053 char heap_remove(HEAP *h, void* val);
00054
00055 #endif // !HEAP_H
```

5.12 controllers/uav_controller/src/util/includes/map.h File Reference

```
#include ¨vec.h¨
#include ¨../../includes/fds_types.h¨
```

Data Structures

- struct Tuple

Macros

- #define **MAP_SIZE** 100

Typedefs

- typedef Cell ∗∗ **Map**

Functions

- Map **map_create** ()
- Cell ∗∗ map_get_cells (Map ∗m, Vec3d v, int ∗num_cells)

  Returns adjacent cells of a given point.
- State ∗∗ **map_get_nbrs** (Map ∗m, State ∗s, int ∗num_nbrs)

  Returns the neighbor of a specific state.
- Tuple ∗ **map_get_connbrs** (Map ∗m, State ∗s, int ∗num_nbrs)

  Returns the contiguous neighbors of a state s.
- Cell ∗∗ map_get_cells_from_states (Map ∗m, State ∗s1, State ∗s2, State ∗s3, int ∗num_cells)

  Finds a specific cell given the adjacent states.
- State ∗ **map_get_state** (Map ∗m, Vec3d v)

  Returns a state given its position as a vector.
- Cell ∗∗ **map_set_cells_cost** (Map ∗m, Vec3d v, double cost, int ∗num_cells)

  Sets the cost of one or more cells given a vector v and the cost. Sets the number of returned cells into num_cells.
- void **map_cleanup** (Map ∗m)

  Cleanup function.

5.12.1 Detailed Description

Map functionality

5.12.2 Function Documentation

map_get_cells()

```
Cell ** map_get_cells (
            Map * m,
            Vec3d v,
            int * num_cells )
```

Returns adjacent cells of a given point.

This function returns one or more cells from a point vector. If the point falls on the edge between two cells both cells will be returned. If the point lies on a vertex of a cell, it will return all four adjacent cells.

map_get_cells_from_states()

```
Cell ** map_get_cells_from_states (
            Map * m,
            State * s1,
            State * s2,
            State * s3,
            int * num_cells )
```

Finds a specific cell given the adjacent states.

Given three distinct states, this function returns a pointer to a Cell adjacent to all three states or NULL if such cell does not exits.

5.13 map.h

Go to the documentation of this file.

```
00001
00007 #include "vec.h"
00008 #include "../../includes/fds_types.h"
00009
00010 #ifndef MAP_H
00011
00012 #define MAP_H
00013 #define MAP_SIZE 100
00014
00015 typedef Cell** Map;
00016
00017 typedef struct {
00018     State* fst;
00019     State* snd;
00020 } Tuple;
00021
00022 Map map_create();
00023
00025
00030 Cell** map_get_cells(Map *m, Vec3d v, int *num_cells);
00031
00033 State** map_get_nbrs(Map *m, State *s, int *num_nbrs);
00034
00036 Tuple* map_get_connbrs(Map *m, State *s, int *num_nbrs);
00037
00039
00042 Cell** map_get_cells_from_states(Map *m, State *s1, State *s2, State *s3, int *num_cells);
00043
00045 State* map_get_state(Map *m, Vec3d v);
00046
00048 Cell** map_set_cells_cost(Map *m, Vec3d v, double cost, int *num_cells);
00049
00050 static inline int states_are_equal(State *s1, State *s2) {
00051     return s1->v.x == s2->v.x && s1->v.y == s2->v.y;
00052 }
00053
00055 void map_cleanup(Map *m);
00056
00057 #endif // !MAP_H
```

5.14 controllers/uav_controller/src/util/includes/util.h File Reference

Functions

- void **init_debug_file** ()

  Initializes the logging library.
- void **cleanup_debug_file** ()

  Cleans up the logging library.
- void **logvi** (int val, char ∗name)

  Write to the log file one integer value.
- void **log2vi** (int val1, char ∗name1, int val2, char ∗name2)

  Write to the log file two integer values.
- void **logvf** (double val, char ∗name)

  Write to the log file one float value.
- void **log2vf** (double val1, char ∗name1, double val2, char ∗name2)

  Write to the log file two float values.
- void **logs** (char ∗str)

  Write a string to the log file.
- double **to_rad** (double alpha)

  Converts an angle in degrees into radians.
- double **to_deg** (double alpha)

  Converts an angle in radians into degrees.
- unsigned char **dtouc** (double d)

  Convert a double value to unsigned character (deprecated)

5.14.1 Detailed Description

A simple utility library

5.15 util.h

Go to the documentation of this file.

```
00001
00006 #ifndef UTIL_H
00007 #define UTIL_H
00008
00009 void init_debug_file();
00010 void cleanup_debug_file();
00011
00012 // Log functions
00013 void logvi(int val, char *name);
00014 void log2vi(int val1, char *name1, int val2, char *name2);
00015
00016 void logvf(double val, char *name);
00017 void log2vf(double val1, char *name1, double val2, char *name2);
00018
00019 void logs(char *str);
00020
00021 double to_rad(double alpha);
00022 double to_deg(double alpha);
00023
00024 unsigned char dtouc(double d);
00025
00026 #endif // !UTIL_H
```

5.16 controllers/uav_controller/src/util/includes/vec.h File Reference

```
#include <math.h>
```

Data Structures

- union Vec3d

Typedefs

- typedef union Vec3d Vec3d

5.16.1 Detailed Description

A simple vector library

5.16.2 Typedef Documentation

Vec3d

```
typedef union Vec3d Vec3d
```

A three-dimensional vector

5.17 vec.h

Go to the documentation of this file.

```
00001
00007 #ifndef VEC_H
00008 #define VEC_H
00009
00010 #include <math.h>
00011
00015 typedef union Vec3d {
00016     struct {
00017         double x;
00018         double y;
00019         double z;
00020     };
00021
00022     struct {
00023         double pitch;
00024         double roll;
00025         double yaw;
00026     };
00027 } Vec3d;
00028
00032 static inline int vec_equal(Vec3d v, Vec3d u) {
00033     return v.x == u.x && v.y == u.y;
00034 }
00035
00039 static inline Vec3d vec_rotate(Vec3d v, double alpha) {
00040     double xp = v.x * cos(alpha) - v.y * sin(alpha);
00041     double yp = v.x * sin(alpha) + v.y * cos(alpha);
00042     Vec3d p = { xp, yp, 0 };
00043     return p;
00044 }
00045
00049 static inline Vec3d vec_translate(Vec3d v, Vec3d u) {
00050     Vec3d pf = { v.x + u.x, v.y + u.y, v.z + u.z };
00051     return pf;
00052 }
00053
00054 #endif // !VEC_H
```

Index